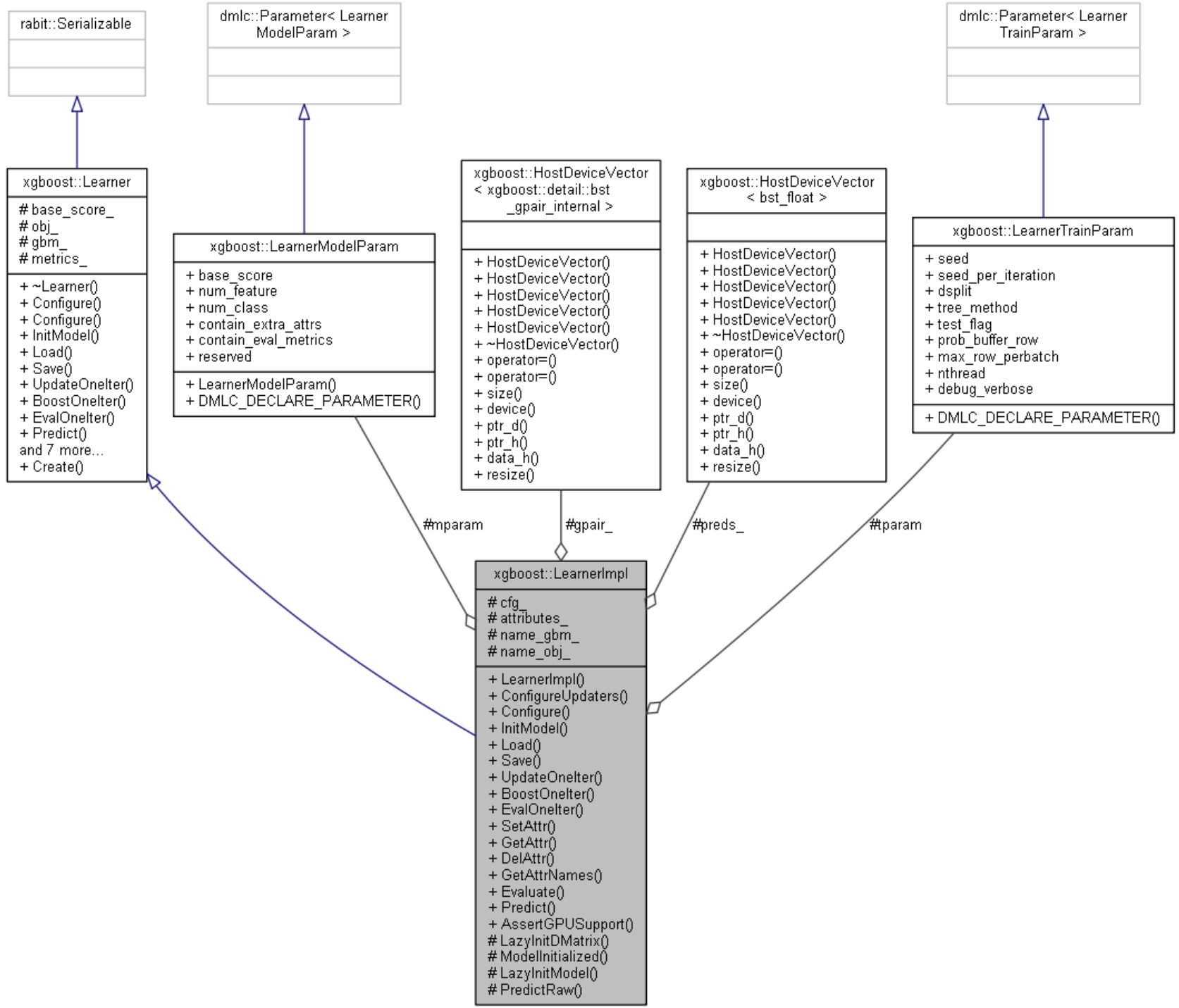
XGBoost in practice

Workflow

```
Cli_main.cc:
main()
    -> CLIRunTask()
        -> CLITrain()
            -> DMatrix::Load()
            -> learner = Learner::Create()
            -> learner->Configure()
            -> learner->InitModel()
            -> for (i = 0; i < param.num_round; ++i) //training starts here
                -> learner->UpdateOneIter()
                -> learner->Save()
```

Learner


rabit::Serializable
dmlc::Parameter< LearnerModelParam >
dmlc::Parameter< LearnerTrainParam >
xgboost::Learner
base_score_
obj_
gbm_
metrics_
+ ~Learner()
+ Configure()
+ Configure()
+ InitModel()
+ Load()
+ Save()
+ UpdateOneIter()
+ BoostOneIter()
+ EvalOneIter()
+ Predict()
and 7 more...
+ Create()
xgboost::LearnerModelParam
+ base_score
+ num_feature
+ num_class
+ contain_extra_attrs
+ contain_eval_metrics
+ reserved
+ LearnerModelParam()
+ DMLC_DECLARE_PARAMETER()
xgboost::HostDeviceVector< xgboost::detail::bst_gpair_internal >
+ HostDeviceVector()
+ HostDeviceVector()
+ HostDeviceVector()
+ HostDeviceVector()
+ HostDeviceVector()
+ ~HostDeviceVector()
+ operator=()
+ operator=()
+ size()
+ device()
+ ptr_d()
+ ptr_h()
+ data_h()
+ resize()
xgboost::HostDeviceVector< bst_float >
+ HostDeviceVector()
+ HostDeviceVector()
+ HostDeviceVector()
+ HostDeviceVector()
+ HostDeviceVector()
+ ~HostDeviceVector()
+ operator=()
+ operator=()
+ size()
+ device()
+ ptr_d()
+ ptr_h()
+ data_h()
+ resize()
xgboost::LearnerTrainParam
+ seed
+ seed_per_iteration
+ dsplit
+ tree_method
+ test_flag
+ prob_buffer_row
+ max_row_perbatch
+ nthread
+ debug_verbose
+ DMLC_DECLARE_PARAMETER()
#mparam
#gpair_
#preds_
#param
xgboost::LearnerImpl
cfg_
attributes_
name_gbm_
name_obj_
+ LearnerImpl()
+ ConfigureUpdaters()
+ Configure()
+ InitModel()
+ Load()
+ Save()
+ UpdateOneIter()
+ BoostOneIter()
+ EvalOneIter()
+ SetAttr()
+ GetAttr()
+ DelAttr()
+ GetAttrNames()
+ Evaluate()
+ Predict()
+ AssertGPUSupport()
# LazyInitDMatrix()
# ModelInitialized()
# LazyInitModel()
# PredictRaw()

Learner Parameters

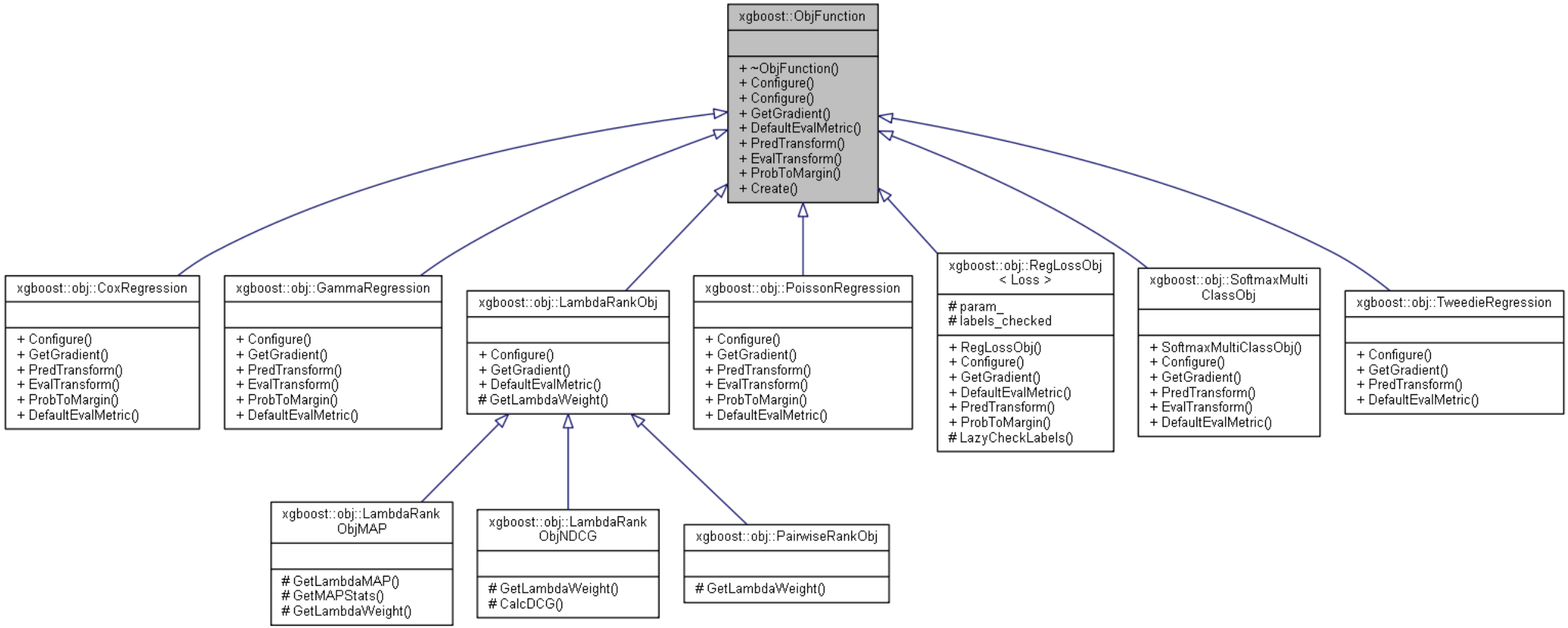
```
LearnerTrainParam:
  int seed; // stored random seed
  bool seed_per_iteration;   // whether seed the PRNG each
iteration
  int dsplit;  // data split mode, can be row, col, or none.
        .set_default(0)
        .add_enum("auto", 0)
        .add_enum("col", 1)
        .add_enum("row", 2)
        .describe("Data split mode for distributed training.");
  int tree_method;  // tree construction method
        .set_default(0)
        .add_enum("auto", 0)
        .add_enum("approx", 1)
        .add_enum("exact", 2)
        .add_enum("hist", 3)
        .add_enum("gpu_exact", 4)
        .add_enum("gpu_hist", 5)
        .describe("Choice of tree construction method.");
  float prob_buffer_row;  // maximum buffered row value
        .set_default(1.0f)
        .set_range(0.0f, 1.0f)
        .describe("Maximum buffered row portion");
  size_t max_row_perbatch;  // maximum row per batch.
  int nthread;  // number of threads to use if OpenMP is
enabled. if equals 0, use system default
          .set_default(0)
          .describe("Number of threads to use.");
```

```
/*! \brief training parameter for regression */
LearnerModelParam:
  /* \brief global bias */
  bst_float base_score;
        .set_default(0.5f)
        .describe("Global bias of the model.");
  /* \brief number of features  */
  unsigned num_feature;
        .set_default(0)
        .describe(
            "Number of features in training data, this parameter
will be automatically detected by learner.");
  /* \brief number of classes, if it is multi-class
classification  */
  int num_class;
          .set_default(0)
          .set_lower_bound(0)
          .describe( "Number of class option for multi-class
classifier. By default equals 0 and corresponds to binary
classifier.");
  /*! \brief Model contain eval metrics */
  int contain_eval_metrics;
```

Main Workflow

```
learner.cc:
Create()
    -> new LearnerImpl() // concrete learner class
Configure()
    1. infer num_class, num_feature
    2. ConfigureUpdaters()
    3. config objectives
InitModel()
    -> LazyInitModel()
        -> obj_ = ObjFunction::Create()
        -> gbm_ = GradientBooster::Create()
        -> obj_->Configure()
        -> gbm_->Configure()
UpdateOneIter()
    -> LazyInitDMatrix() //sort features, CSR->CSC
    -> PredictRaw() //get y_hat using F_t-1
    -> obj_->GetGradient() // get g and h
    -> gbm_->DoBoost() // boost new tree
```

Objective


xgboost::ObjFunction
+ ~ObjFunction()
+ Configure()
+ Configure()
+ GetGradient()
+ DefaultEvalMetric()
+ PredTransform()
+ EvalTransform()
+ ProbToMargin()
+ Create()
xgboost::obj::CoxRegression
+ Configure()
+ GetGradient()
+ PredTransform()
+ EvalTransform()
+ ProbToMargin()
+ DefaultEvalMetric()
xgboost::obj::GammaRegression
+ Configure()
+ GetGradient()
+ PredTransform()
+ EvalTransform()
+ ProbToMargin()
+ DefaultEvalMetric()
xgboost::obj::LambdaRankObj
+ Configure()
+ GetGradient()
+ DefaultEvalMetric()
# GetLambdaWeight()
xgboost::obj::PoissonRegression
+ Configure()
+ GetGradient()
+ PredTransform()
+ EvalTransform()
+ ProbToMargin()
+ DefaultEvalMetric()
xgboost::obj::RegLossObj < Loss >
param_
labels_checked
+ RegLossObj()
+ Configure()
+ GetGradient()
+ DefaultEvalMetric()
+ PredTransform()
+ ProbToMargin()
# LazyCheckLabels()
xgboost::obj::SoftmaxMulti ClassObj
+ SoftmaxMultiClassObj()
+ Configure()
+ GetGradient()
+ PredTransform()
+ EvalTransform()
+ DefaultEvalMetric()
xgboost::obj::TweedieRegression
+ Configure()
+ GetGradient()
+ PredTransform()
+ DefaultEvalMetric()
xgboost::obj::LambdaRank ObjMAP
GetLambdaMAP()
GetMAPStats()
GetLambdaWeight()
xgboost::obj::LambdaRank ObjNDCG
GetLambdaWeight()
CalcDCG()
xgboost::obj::PairwiseRankObj
GetLambdaWeight()

Objective

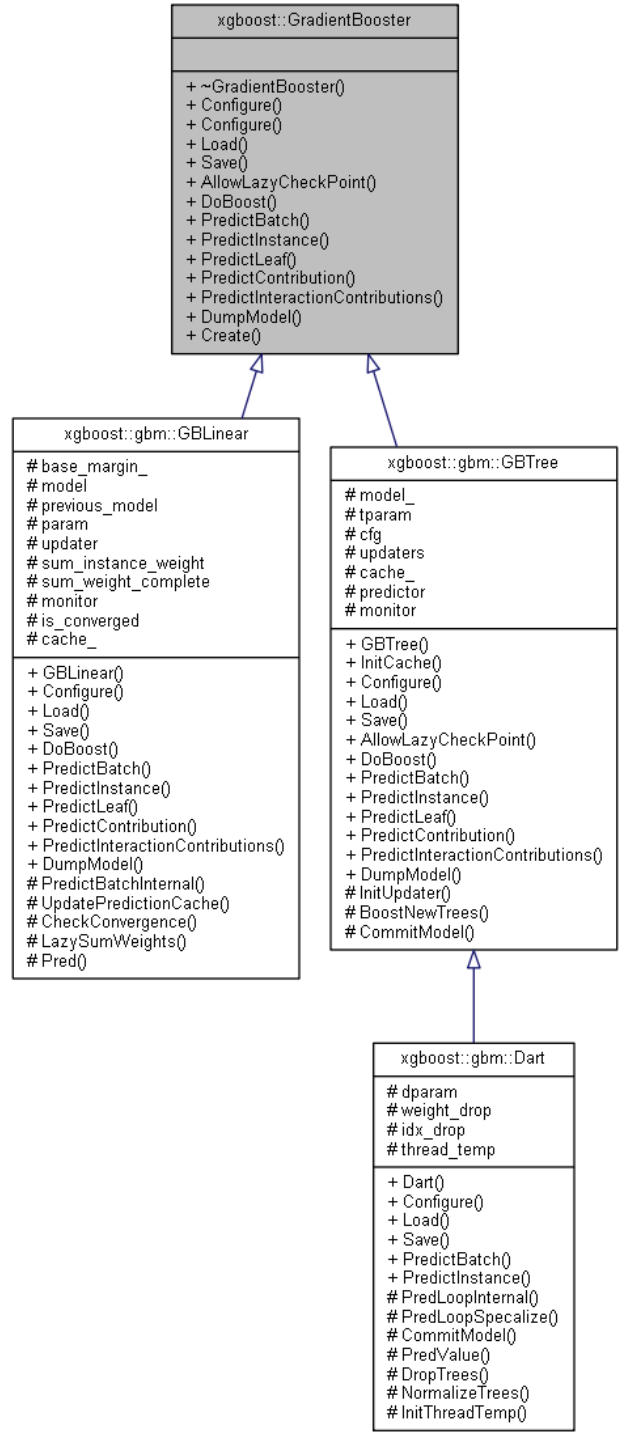
- Can customize objective function, only need to provide 1st and 2nd order gradient formula

```
SoftmaxMultiClass, "multi:softmax"
    SoftmaxMultiClassObj(output_prob=false)
SoftprobMultiClass, "multi:softprob"
    SoftmaxMultiClassObj(output_prob=true)
    loss function is mlogloss(multi-class logloss)
PairwiseRankObj, "rank:pairwise"
LambdaRankNDCG, "rank:ndcg"
LambdaRankObjMAP, "rank:map"
LinearRegression, "reg:linear"
    RegLossObj<LinearSquareLoss>, eval=rmse
LogisticRegression, "reg:logistic"
    RegLossObj<LogisticRegression>, eval=rmse
LogisticClassification, "binary:logistic"
    RegLossObj<LogisticClassification>, eval=error
LogisticRaw, "binary:logitraw"
    RegLossObj<LogisticRaw>, eval=auc
PoissonRegression, "count:poisson"
CoxRegression, "survival:cox"
GammaRegression, "reg:gamma"
TweedieRegression, "reg:tweedie"
GPULinearRegression, "gpu:reg:linear"
GPULogisticRegression, "gpu:reg:logistic"
GPULogisticClassification, "gpu:binary:logistic"
GPULogisticRaw, "gpu:binary:logitraw"
```

Main Workflow

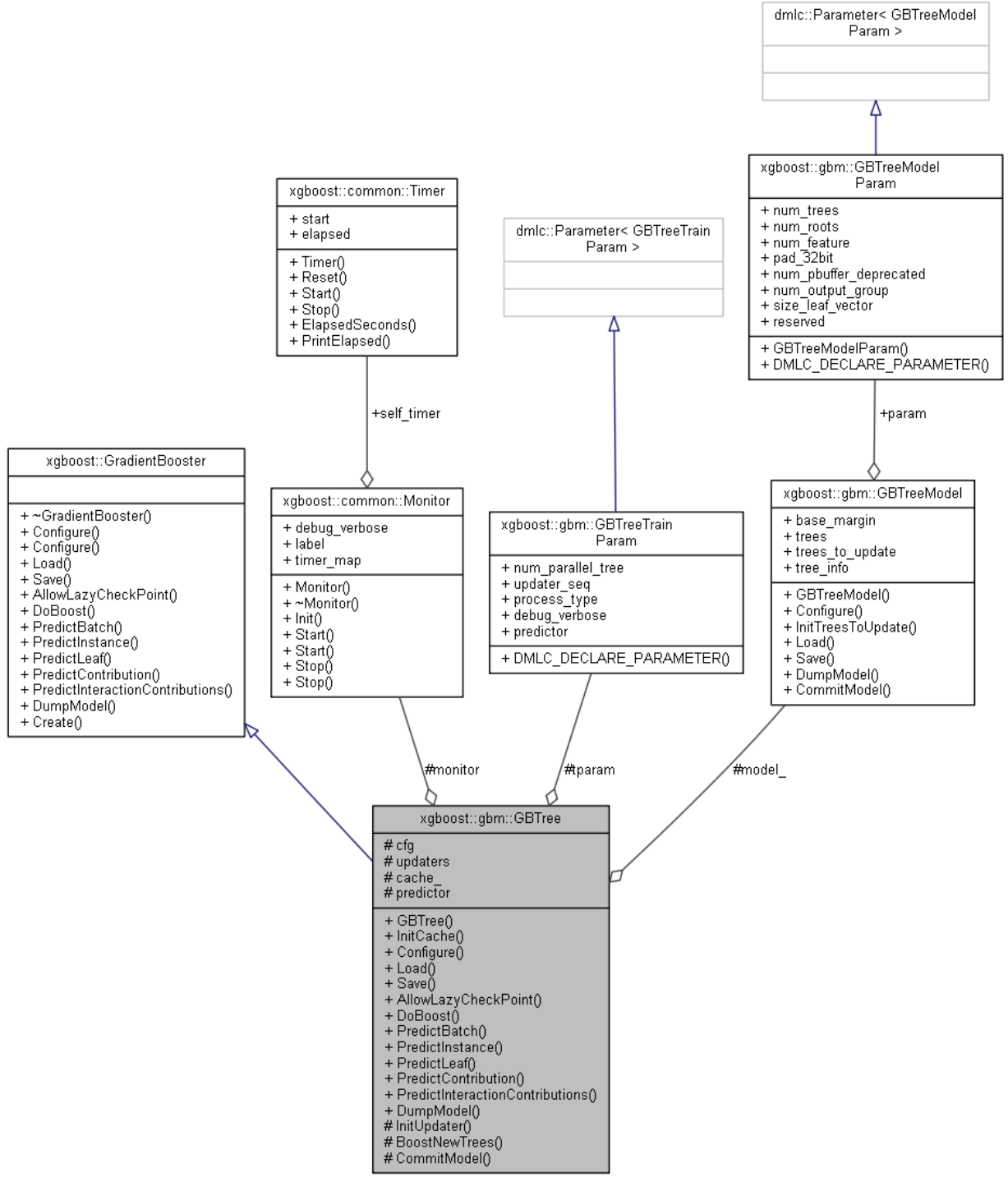
```
learner.cc:
Create()
    -> new LearnerImpl() // concrete learner class
Configure()
    1. infer num_class, num_feature
    2. ConfigureUpdaters()
    3. config objectives
InitModel()
    -> LazyInitModel()
        -> obj_ = ObjFunction::Create()
        -> gbm_ = GradientBooster::Create()
        -> obj_->Configure()
        -> gbm_->Configure()
UpdateOneIter()
    -> LazyInitDMatrix() //sort features, CSR->CSC
    -> PredictRaw() //get y_hat using F_t-1
    -> obj_->GetGradient() // get g and h
    -> gbm_->DoBoost() // boost new tree
```

GBM


xgboost::GradientBooster
+ ~GradientBooster()
+ Configure()
+ Configure()
+ Load()
+ Save()
+ AllowLazyCheckPoint()
+ DoBoost()
+ PredictBatch()
+ PredictInstance()
+ PredictLeaf()
+ PredictContribution()
+ PredictInteractionContributions()
+ DumpModel()
+ Create()
xgboost::gbm::GBLinear
base_margin_
model
previous_model
param
updater
sum_instance_weight
sum_weight_complete
monitor
is_converged
cache_
+ GBLinear()
+ Configure()
+ Load()
+ Save()
+ DoBoost()
+ PredictBatch()
+ PredictInstance()
+ PredictLeaf()
+ PredictContribution()
+ PredictInteractionContributions()
+ DumpModel()
# PredictBatchInternal()
# UpdatePredictionCache()
# CheckConvergence()
# LazySumWeights()
# Pred()
xgboost::gbm::GBTree
model_
tparam
cfg
updaters
cache_
predictor
monitor
+ GBTree()
+ InitCache()
+ Configure()
+ Load()
+ Save()
+ AllowLazyCheckPoint()
+ DoBoost()
+ PredictBatch()
+ PredictInstance()
+ PredictLeaf()
+ PredictContribution()
+ PredictInteractionContributions()
+ DumpModel()
# InitUpdater()
# BoostNewTrees()
# CommitModel()
xgboost::gbm::Dart
dparam
weight_drop
idx_drop
thread_temp
+ Dart()
+ Configure()
+ Load()
+ Save()
+ PredictBatch()
+ PredictInstance()
# PredLoopInternal()
# PredLoopSpecalize()
# CommitModel()
# PredValue()
# DropTrees()
# NormalizeTrees()
# InitThreadTemp()

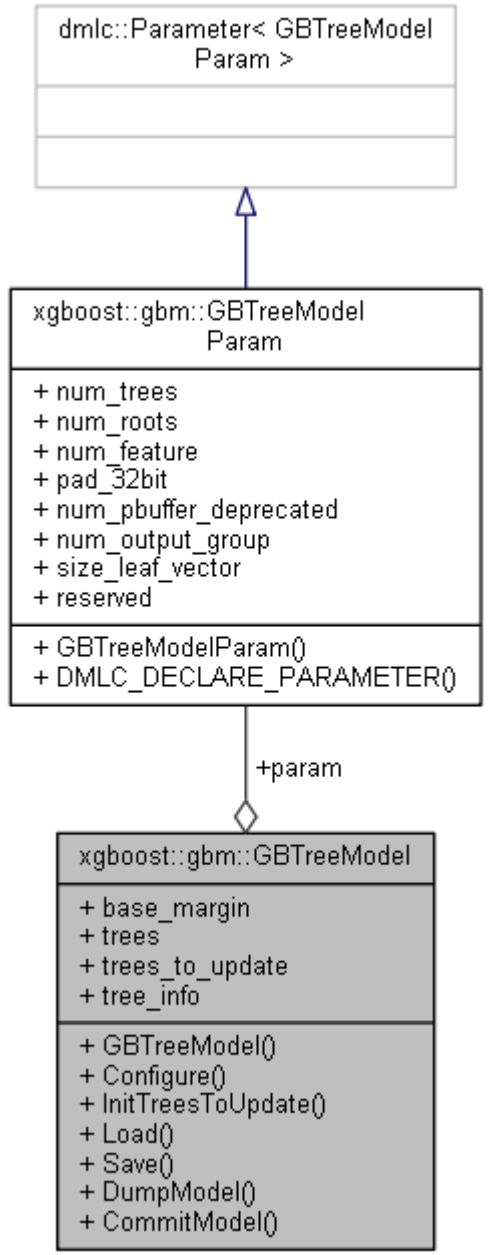
GBTree

```
/*! \brief training parameters */
GBTreeTrainParam
  /*!
   * \brief number of parallel trees constructed each iteration
   *  use this option to support boosted random forest
   */
  int num_parallel_tree;
        .set_default(1)
        .set_lower_bound(1)
        .describe("Number of parallel trees constructed during
each iteration. This option is used to support boosted random
forest");
  /*! \brief tree updater sequence */
  std::string updater_seq/updater;
        .set_default("grow_colmaker,prune")
        .describe("Tree updater sequence.");
  /*! \brief type of boosting process to run */
  int process_type;
        .set_default(kDefault)
        .add_enum("default", kDefault) //create new trees
        .add_enum("update", kUpdate) //update trees in existing
model
        .describe("Whether to run the normal boosting process
that creates new trees, or to update the trees in an existing
model.");
  std::string predictor;
      .set_default("cpu_predictor")
      .describe("Predictor algorithm type");
```



GBTreeModel

```
GBTreeModel
  // base margin
  bst_float base_margin;
  // model parameter
  GBTreeModelParam param;
  /*! \brief vector of trees stored in the model */
  std::vector<std::unique_ptr<RegTree> > trees;
  /*! \brief for the update process, a place to keep the initial trees */
  std::vector<std::unique_ptr<RegTree> > trees_to_update;
  /*! \brief some information indicator of the tree, reserved */
  std::vector<int> tree_info;
```



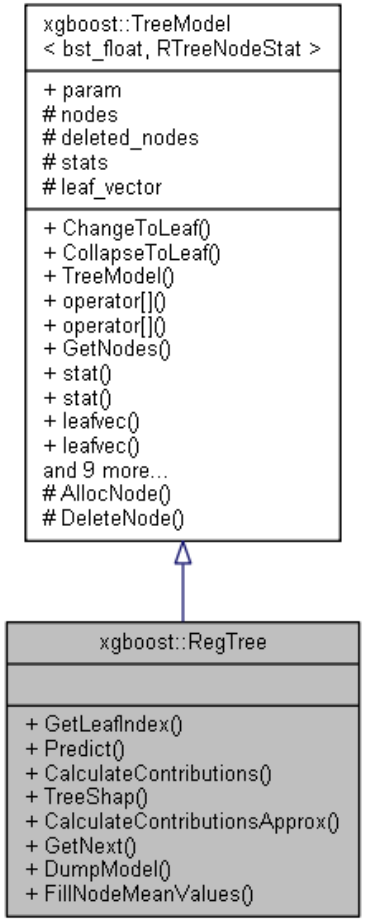
RegTree

TreeModel, RegTree:

```
// vector of nodes
std::vector<Node> nodes;
// free node space, used during training process
std::vector<int>  deleted_nodes;
// stats of nodes
std::vector<TNodeStat> stats;
```

TreeParam

```
/*! \brief number of start root */
int num_roots;
/*! \brief total number of nodes */
int num_nodes;
/*!\brief number of deleted nodes */
int num_deleted;
/*! \brief maximum depth, this is a statistics of the tree */
int max_depth;
/*! \brief number of features used for tree construction */
int num_feature;
```



Node

Node:

```
member value default=0
// pointer to parent, highest bit is used to
// indicate whether it's a left child or not
int parent_; //if parent_ == -1, it’s root node
// pointer to left, right
int cleft_, cright_; //if cleft_==-1, is_leaf=true
// split feature index, left split or right split depends on
the highest bit
unsigned sindex_; //if sindex_==MAX_INT, means this
node is marked deleted
// extra info
Info info_; {float leaf_value, TSplitCond split_cond}
```

Gradient Stats

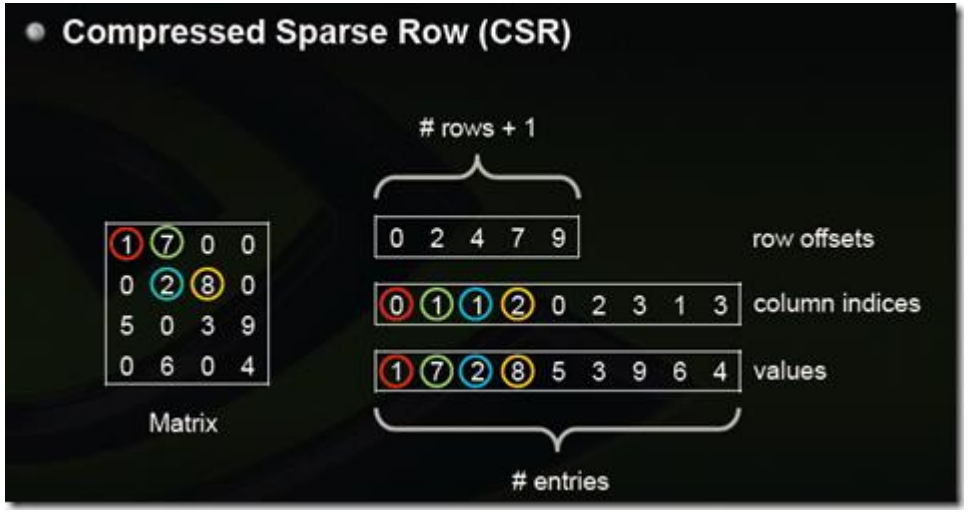
```
/*! \brief core statistics used for tree construction */
GradStats
 /*! \brief sum gradient statistics */
 double sum_grad;
 /*! \brief sum hessian statistics */
 double sum_hess;
 /*! \brief add statistics to the data */
 inline void Add(const GradStats& b) {
  sum_grad += b.sum_grad;
  sum_hess += b.sum_hess;
 }
```

Implemented **CalcWeight, CalcGain, CalcSplitGain**

Main Workflow

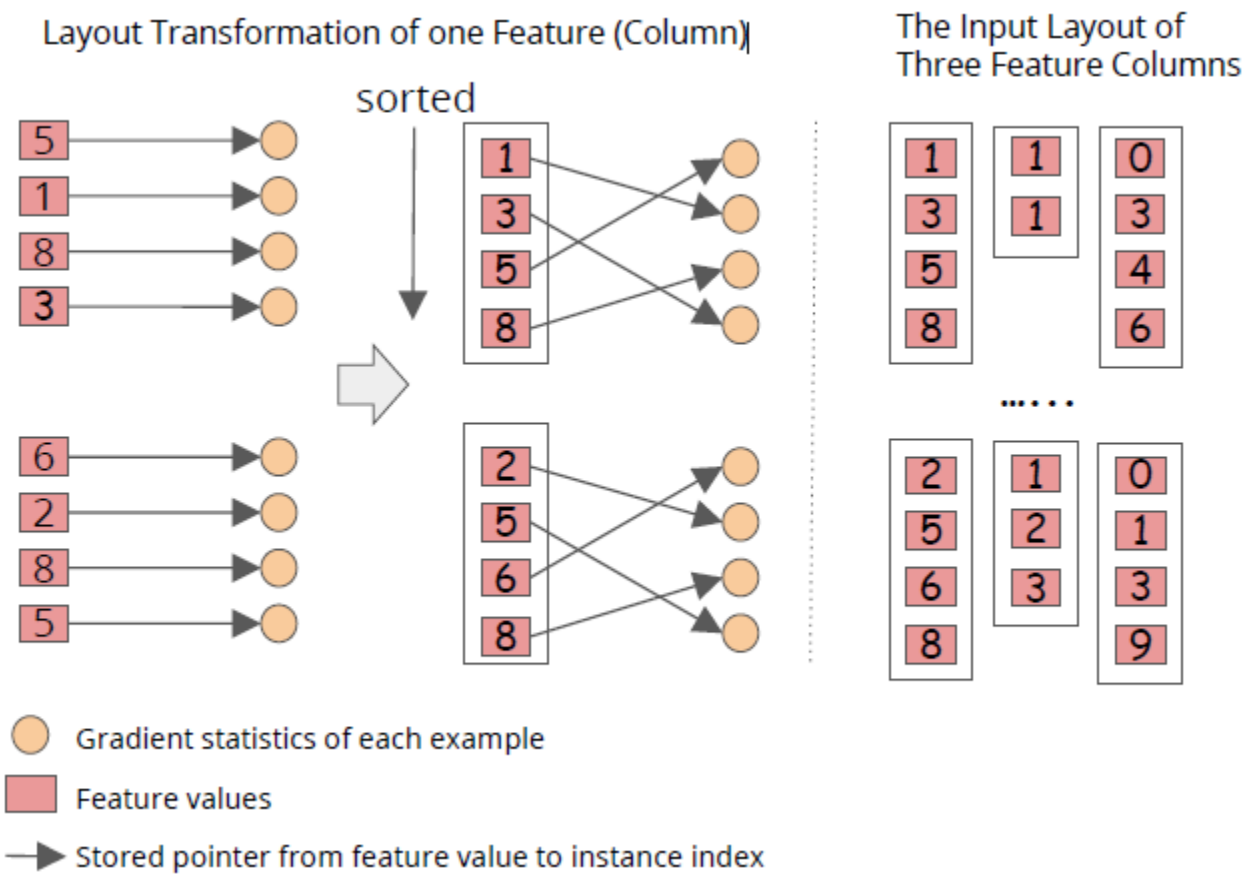
```
learner.cc:
Create()
    -> new LearnerImpl() // concrete learner class
Configure()
    1. infer num_class, num_feature
    2. ConfigureUpdaters()
    3. config objectives
InitModel()
    -> LazyInitModel()
        -> obj_ = ObjFunction::Create()
        -> gbm_ = GradientBooster::Create()
        -> obj_->Configure()
        -> gbm_->Configure()
UpdateOneIter()
    -> LazyInitDMatrix() //sort features, CSR->CSC
    -> PredictRaw() //get y_hat using F_t-1
    -> obj_->GetGradient() // get g and h
    -> gbm_->DoBoost() // boost new tree
```

CSR and CSC



- Corresponding CSC representation
- Values： [1 5 7 2 6 8 3 9 4]
- Row Indices：[0 2 0 1 3 1 2 2 3]
- Column Offsets：[0 2 5 7 9]
- First read data as CSR, then parallel transform & sort to CSC
- This is done in “LazyInitDMatrix”

The Column based Input Block



Main Workflow

```
learner.cc:
Create()
    -> new LearnerImpl() // concrete learner class
Configure()
    1. infer num_class, num_feature
    2. ConfigureUpdaters()
    3. config objectives
InitModel()
    -> LazyInitModel()
        -> obj_ = ObjFunction::Create()
        -> gbm_ = GradientBooster::Create()
        -> obj_->Configure()
        -> gbm_->Configure()
UpdateOneIter()
    -> LazyInitDMatrix() //sort features, CSR->CSC
    -> PredictRaw() //get y_hat using F_t-1
    -> obj_->GetGradient() // get g and h
    -> gbm_->DoBoost() // boost new tree
```

PredictRaw

```
learner.cc
PredictRaw() // use F_t-1 to get y_hat
    ->gbm_->PredictBatch()
        ->predictor->PredictBatch()
            ->PredLoopInternal()
                ->predloopspecalize()
                    -> for inst in batch:
                        -> PredValue(inst)
```

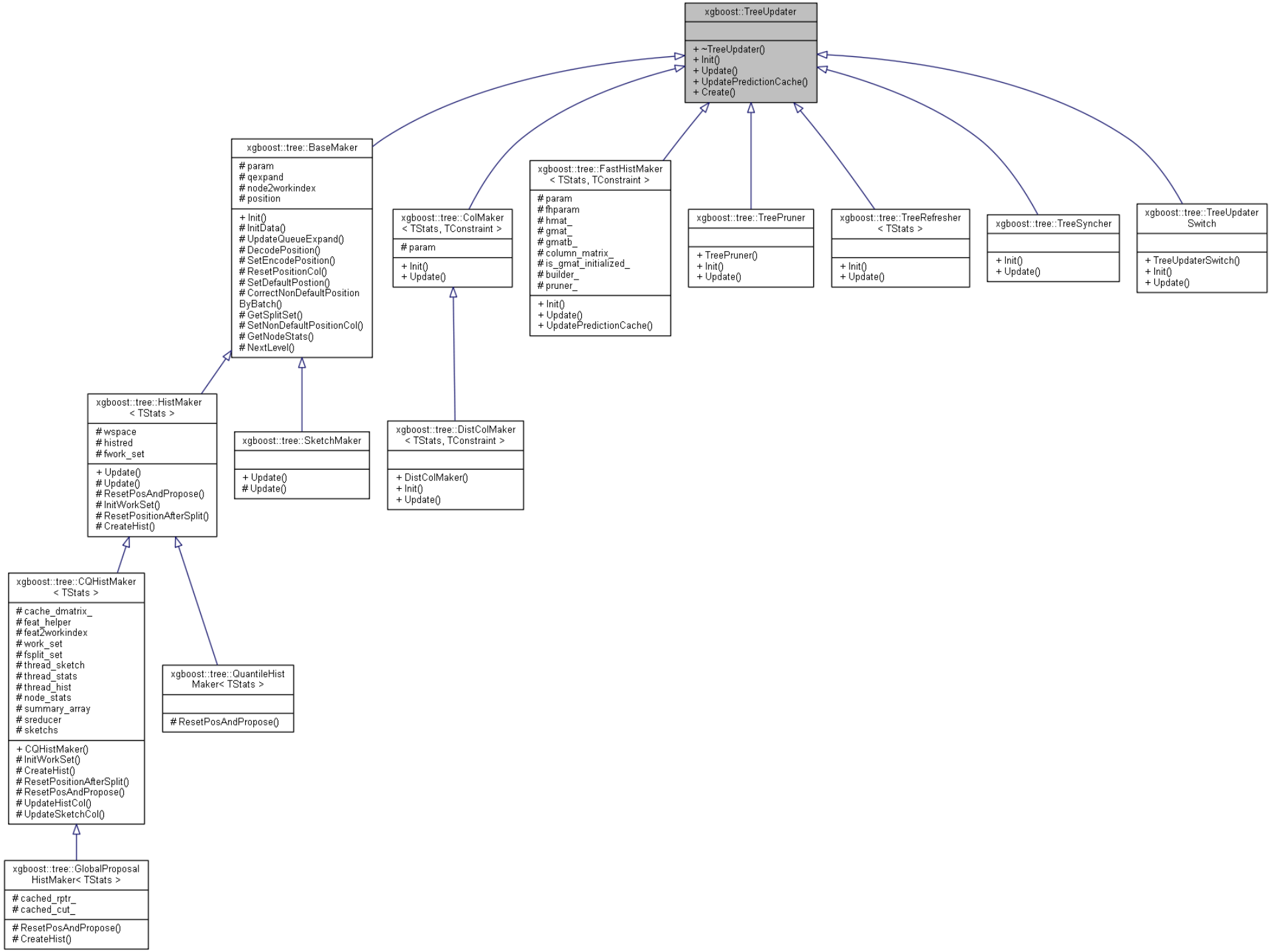
Main Workflow

```
learner.cc:
Create()
    -> new LearnerImpl() // concrete learner class
Configure()
    1. infer num_class, num_feature
    2. ConfigureUpdaters()
    3. config objectives
InitModel()
    -> LazyInitModel()
        -> obj_ = ObjFunction::Create()
        -> gbm_ = GradientBooster::Create()
        -> obj_->Configure()
        -> gbm_->Configure()
UpdateOneIter()
    -> LazyInitDMatrix() //sort features, CSR->CSC
    -> PredictRaw() //get y_hat using F_t-1
    -> obj_->GetGradient() // get g and h
    -> gbm_->DoBoost() // boost new tree
```

GBTree cont’d

```
gbtree.cc:
Configure() // create all updates according to updater_seq
      -> for (up in updaters)
            -> up->Init()
DoBoost() // grow new tree
      -> BoostNewTrees()
            -> new_tree = new RegTree()
            -> for (up in updaters)
                  -> up->Update(new_tree)
```

Updaters


xgboost::TreeUpdater
+ ~TreeUpdater()
+ Init()
+ Update()
+ UpdatePredictionCache()
+ Create()
xgboost::tree::BaseMaker
param
qexpand
node2workindex
position
+ Init()
# InitData()
# UpdateQueueExpand()
# DecodePosition()
# SetEncodePosition()
# ResetPositionCol()
# SetDefaultPostion()
# CorrectNonDefaultPosition ByBatch()
# GetSplitSet()
# SetNonDefaultPositionCol()
# GetNodeStats()
# NextLevel()
xgboost::tree::ColMaker < TStats, TConstraint >
param
+ Init()
+ Update()
xgboost::tree::FastHistMaker < TStats, TConstraint >
param
fhparam
hmat_
gmat_
gmatb_
column_matrix_
is_gmat_initialized_
builder_
pruner_
+ Init()
+ Update()
+ UpdatePredictionCache()
xgboost::tree::TreePruner
+ TreePruner()
+ Init()
+ Update()
xgboost::tree::TreeRefresher < TStats >
+ Init()
+ Update()
xgboost::tree::TreeSyncher
+ Init()
+ Update()
xgboost::tree::TreeUpdater Switch
+ TreeUpdaterSwitch()
+ Init()
+ Update()
xgboost::tree::HistMaker < TStats >
wspace
histred
fwork_set
+ Update()
# Update()
# ResetPosAndPropose()
# InitWorkSet()
# ResetPositionAfterSplit()
# CreateHist()
xgboost::tree::SketchMaker
+ Update()
# Update()
xgboost::tree::DistColMaker < TStats, TConstraint >
+ DistColMaker()
+ Init()
+ Update()
xgboost::tree::CQHistMaker < TStats >
cache_dmatrix_
feat_helper
feat2workindex
work_set
fsplit_set
thread_sketch
thread_stats
thread_hist
node_stats
summary_array
sreducer
sketchs
+ CQHistMaker()
# InitWorkSet()
# CreateHist()
# ResetPositionAfterSplit()
# ResetPosAndPropose()
# UpdateHistCol()
# UpdateSketchCol()
xgboost::tree::QuantileHist Maker< TStats >
ResetPosAndPropose()
xgboost::tree::GlobalProposal HistMaker< TStats >
cached_rptr_
cached_cut_
ResetPosAndPropose()
CreateHist()

Main Workflow

```
learner.cc:
Create()
    -> new LearnerImpl() // concrete learner class
Configure()
    1. infer num_class, num_feature
    2. ConfigureUpdaters()
    3. config objectives
InitModel()
    -> LazyInitModel()
        -> obj_ = ObjFunction::Create()
        -> gbm_ = GradientBooster::Create()
        -> obj_->Configure()
        -> gbm_->Configure()
UpdateOneIter()
    -> LazyInitDMatrix() //sort features, CSR->CSC
    -> PredictRaw() //get y_hat using F_t-1
    -> obj_->GetGradient() // get g and h
    -> gbm_->DoBoost() // boost new tree
```

Tree Updaters/Booster

```
TreeUpdater:
ColMaker, "grow_colmaker"
DistColMaker, "distcol"
FastHistMaker, "grow_fast_histmaker"
GPUMaker, "grow_gpu"
GPUHistMaker, "grow_gpu_hist"
LocalHistMaker(CQHistMaker), "grow_local_histmaker"
GlobalHistMaker(GlobalProposalHistMaker), "grow_global_histmaker"
HistMaker(GlobalProposalHistMaker), "grow_histmaker"
TreePruner, "prune"
TreeRefresher, "refresh"
SketchMaker, "grow_skmaker"
TreeSyncher, "sync"
```

```
learner.cc:157:ConfigureUpdaters():
    tree_method in ['auto', 'approx', 'exact']:
        dsplit == 'col' -> 'distcol'
        dsplit == 'row' -> 'grow_histmaker,prune'
        prob_buffer_row != 1.0f -> 'grow_histmaker,refresh,prune'
    tree_method == 'hist': -> 'grow_fast_histmaker'
        https://github.com/dmlc/xgboost/issues/1950
    tree_method == 'gpu_exact' -> 'grow_gpu,prune'
        predictor if not specified -> 'gpu_predictor'
    tree_method == 'gpu_hist' -> 'grow_gpu_hist'
        predictor if not specified -> 'gpu_predictor'
learner.cc:472:LazyInitDMatrix()
    tree_method == 'auto' && num_row >= 2^22 -> tree_method='approx'
    safe_max_row=min(2^16, max_row_perbatch) // cap batch size cap to 2^16
    tree_method == 'approx' -> 'grow_histmaker,prune'
```

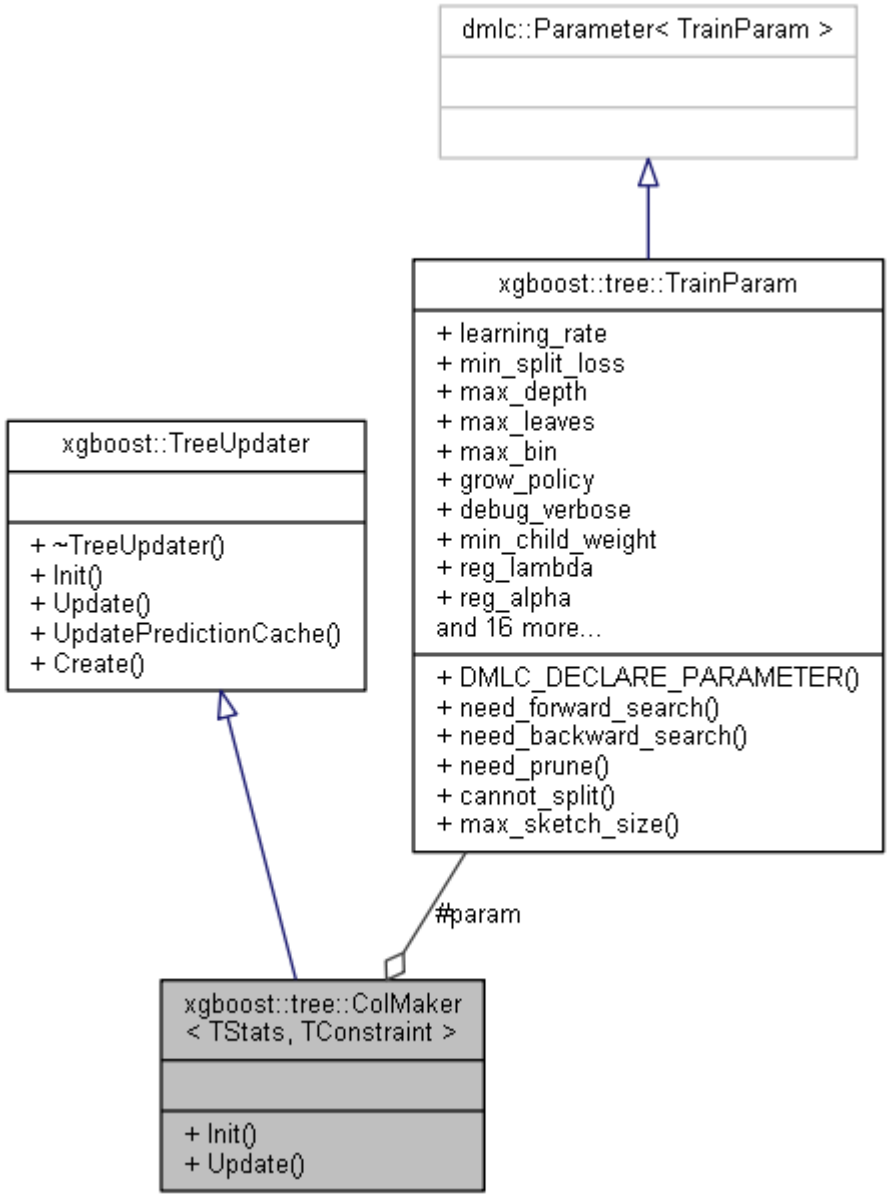
Tree Updaters/Booster Parameters

Updater	**Parameters supported**	**Note**
grow_colmaker/distcol	subsample, max_depth, colsample_bytree, colsample_bylevel, default_direction	Support both colsample_bytree and colsample_bylevel
prune	min_split_loss	
grow_skmaker	subsample, max_depth, sketch_eps	No colsample support
grow_local_histmaker	subsample, max_depth, sketch_eps, colsample_bytree	Only support colsample_bytree
grow_histmaker (grow_global_histmaker)	subsample, max_depth, sketch_eps, colsample_bytree	Only support colsample_bytree
grow_fast_histmaker	subsample, max_bin, max_depth, max_leaves, grow_policy, colsample_bytree	Only support colsample_bytree
All Shared	learning_rate, min_child_weight, reg_lambda, reg_alpha, max_delta_step, num_parallel_tree	All updater support random forest

Updaters support distribution settings:
grow_local_histmaker, grow_histmaker, grow_skmaker, distcol, sync, prune

Colmaker

```
/*! \brief training parameters for regression tree */
TrainParam
  // learning step size for a time
  float learning_rate/eta;
        .set_lower_bound(0.0f)
        .set_default(0.3f)
        .describe("Learning rate(step size) of update.");
  // minimum loss change required for a split
  float min_split_loss/gamma;
        .set_lower_bound(0.0f)
        .set_default(0.0f)
        .describe("Minimum loss reduction required to make a further partition.");
  // maximum depth of a tree
  int max_depth;
        .set_lower_bound(0)
        .set_default(6)
        .describe("Maximum depth of the tree; 0 indicates no limit; a limit is required "
                  "for depthwise policy");
  // maximum number of leaves
  int max_leaves;
            .set_lower_bound(0)
            .set_default(0)
            .describe("Maximum number of leaves; 0 indicates no limit.");
  // if using histogram based algorithm, maximum number of bins per feature
  int max_bin;
            .set_lower_bound(2)
            .set_default(256)
            .describe("if using histogram-based algorithm, maximum number of bins per feature");
  // growing policy
  enum TreeGrowPolicy { kDepthWise = 0, kLossGuide = 1 };
  int grow_policy;
        .set_default(kDepthWise)
        .add_enum("depthwise", kDepthWise)
        .add_enum("lossguide", kLossGuide)
        .describe(
            "Tree growing policy. 0: favor splitting at nodes closest to the node, "
            "i.e. grow depth-wise. 1: favor splitting at nodes with highest loss "
            "change. (cf. LightGBM)");
```



Colmaker

TrainParam cont'd

```
  //----- the rest parameters are less important ----
  // minimum amount of hessian(weight) allowed in a child
  float min_child_weight;
        .set_lower_bound(0.0f).set_default(1.0f)
        .describe("Minimum sum of instance weight(hessian) needed in a child.");
  // L2 regularization factor
  float reg_lambda/lambda;
        .set_lower_bound(0.0f).set_default(1.0f)
        .describe("L2 regularization on leaf weight");
  // L1 regularization factor
  float reg_alpha/alpha;
        .set_lower_bound(0.0f).set_default(0.0f)
        .describe("L1 regularization on leaf weight");
  // default direction choice
  int default_direction;
        .set_default(0).add_enum("learn", 0).add_enum("left", 1).add_enum("right", 2)
        .describe("Default direction choice when encountering a missing value");
            // if default right(all missing value go to large side) then do forward_search(from small to large)
            // if default left, then do backward_search(from large to small). check updater_colmaker.cc:630
  // maximum delta update we can add in weight estimation
  // this parameter can be used to stabilize update
  // default=0 means no constraint on weight delta
  float max_delta_step;
        .set_lower_bound(0.0f).set_default(0.0f)
        .describe("Maximum delta step we allow each tree's weight estimate to be. "\
                  "If the value is set to 0, it means there is no constraint");
  // whether we want to do subsample
  float subsample;
        .set_range(0.0f, 1.0f).set_default(1.0f)
        .describe("Row subsample ratio of training instance.");
  // whether to subsample columns each split, in each level
  float colsample_bylevel;
        .set_range(0.0f, 1.0f).set_default(1.0f)
        .describe("Subsample ratio of columns, resample on each level.");
  // whether to subsample columns during tree construction
  float colsample_bytree;
        .set_range(0.0f, 1.0f).set_default(1.0f)
        .describe("Subsample ratio of columns, resample on each tree construction.");
  // accuracy of sketch
  float sketch_eps;
        .set_range(0.0f, 1.0f).set_default(0.03f)
        .describe("EXP Param: Sketch accuracy of approximate algorithm.");
```

Colmaker::Builder

```
Builder:
    //  --data fields--
    const TrainParam& param;
    // number of omp thread used during training
    const int nthread;
    // Per feature: shuffle index of each feature index
    std::vector<bst_uint> feat_index;
    // Instance Data: current node position in the tree of each instance
    std::vector<int> position;
    // PerThread x PerTreeNode: statistics for per thread construction
    std::vector< std::vector<ThreadEntry> > stemp;
    /*! \brief TreeNode Data: statistics for each constructed node */
    std::vector<NodeEntry> snode;
    /*! \brief queue of nodes to be expanded */
    std::vector<int> qexpand_;
    // constraint value
    std::vector<TConstraint> constraints_;
```

```
ThreadEntry:
    /*! \brief statistics of data */
    TStats stats;
    /*! \brief extra statistics of data */
    TStats stats_extra;
    /*! \brief last feature value scanned */
    bst_float last_fvalue;
    /*! \brief first feature value scanned */
    bst_float first_fvalue;
    /*! \brief current best solution */
    SplitEntry best;
```

```
ColMaker, NodeEntry
    /*! \brief statics for node entry */
    TStats stats;
    /*! \brief loss of this node, without split */
    bst_float root_gain;
    /*! \brief weight calculated related to current data */
    bst_float weight;
    /*! \brief current best solution */
    SplitEntry best;
```

Colmaker::Builder

```
ThreadEntry:
    /*! \brief statistics of data */
    TStats stats;
    /*! \brief extra statistics of data */
    TStats stats_extra;
    /*! \brief last feature value scanned */
    bst_float last_fvalue;
    /*! \brief first feature value scanned */
    bst_float first_fvalue;
    /*! \brief current best solution */
    SplitEntry best;
```

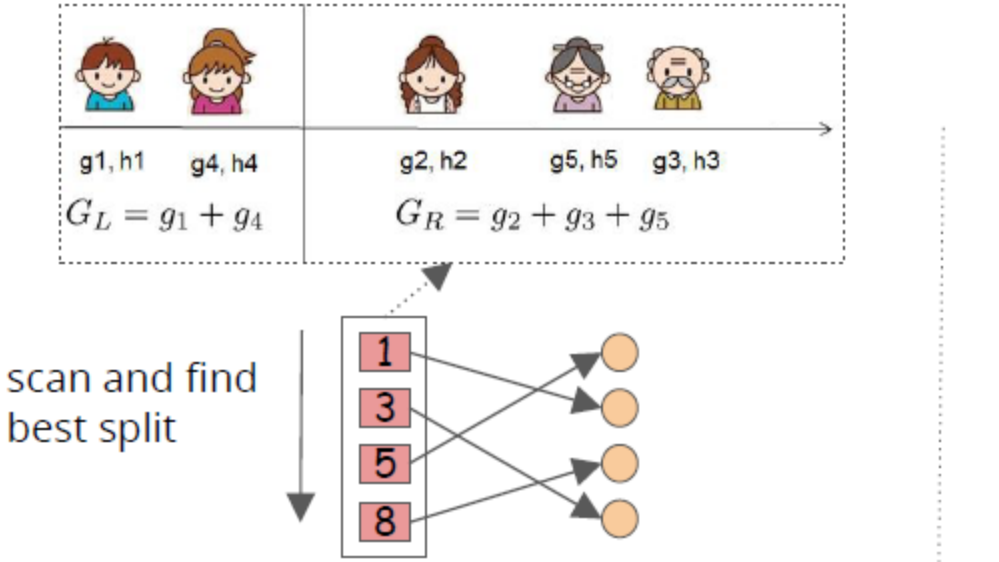
```
ColMaker, NodeEntry
    /*! \brief statics for node entry */
    TStats stats;
    /*! \brief loss of this node, without split */
    bst_float root_gain;
    /*! \brief weight calculated related to current data */
    bst_float weight;
    /*! \brief current best solution */
    SplitEntry best;
```

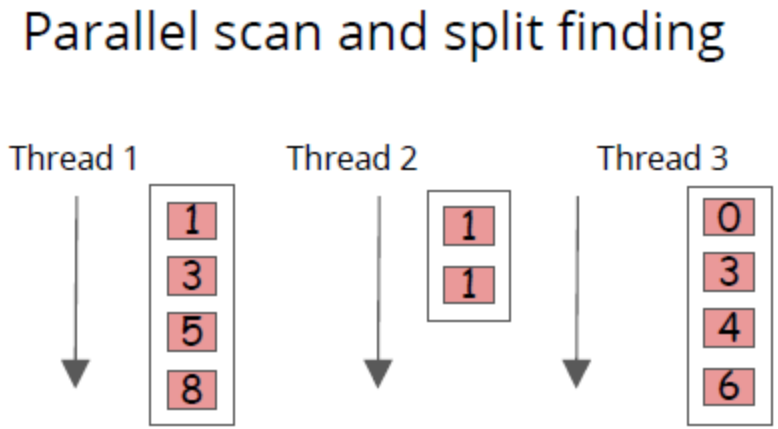
```
/*!
 * \brief statistics that is helpful to store
 *   and represent a split solution for the tree
 */
SplitEntry
  /*! \brief loss change after split this node */
  bst_float loss_chg; //the loss_chg is gain, the large the better
  /*! \brief split index */
  unsigned sindex; //the split index also recored default direction,
follow same convention as in Node
  /*! \brief split value */
  bst_float split_value;

Implemented Update, NeedReplace methods
```

Colmaker::Builder Workflow

```
Builder.Update()
->InitData()
    1. setup position: set every instance belongs to which tree node, init as root=0
    2. do row subsample, bernoulli trial drop instances: set position to  < 0
    3. do colsample_bytree, permutate feature index, only select a subset of feat_index
    4. init stemp to 0
    5. init qexpand_ to root only
->InitNewNode()
    1. for each instance, calc stats and then accumulate result in snode(parallel)
    2. for each node in snode, calc weight&root_gain using stats
->FindSplit()
    1. do colsample_bylevel, permutate feature_index, select a subset as feature_set
    2. for each col batch, UpdateSolution()
        parallel by feature, select the best split per node per feature
        do this twice: default left/right each once
        cache-aware optimization: EnumerateSplitCacheOpt()
    3. SyncBestSolution()
        Use found per node per feature best split, aggregate all feature to get per node best split
        Syncing results from each thread. Because each feature split finding was done in each thread
    4. After sync, create new node base on best split
->ResetPosition()
    recalc all instance's position correspond to new tree structure
->UpdateQueueExpand()
    rebuilt qexpand_, substitute with new nodes
    if no new node, then whole Update process will stop
```

Parallel Split Finding on the Input Layout



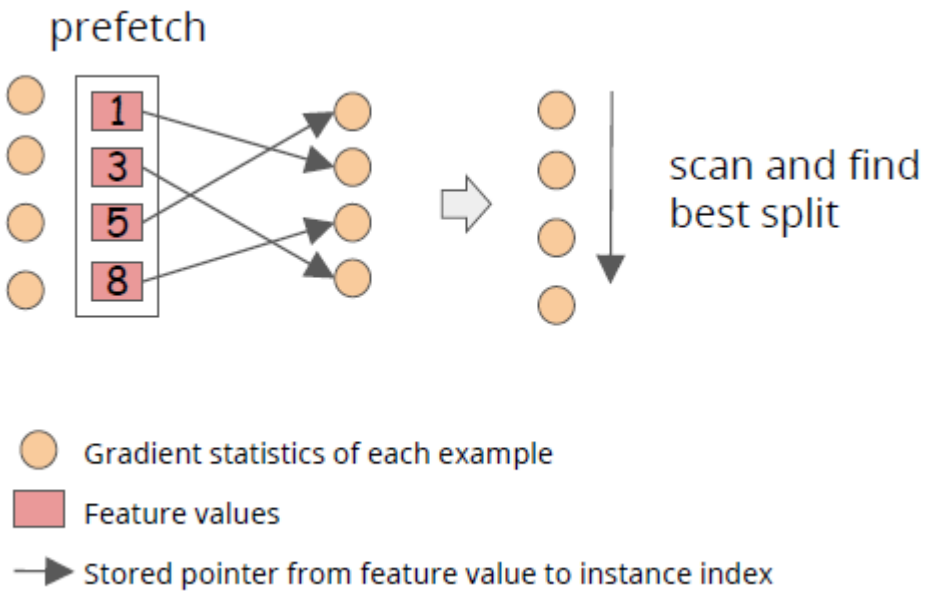


Gradient statistics of each example

Feature values

Stored pointer from feature value to instance index

Cache-aware Prefetching




```
bufg[1] = g[ptr[1]]
bufg[2] = g[ptr[2]]
...

G = G +  bufg[1]

calculate score ...

G = G + bufg[2]
```

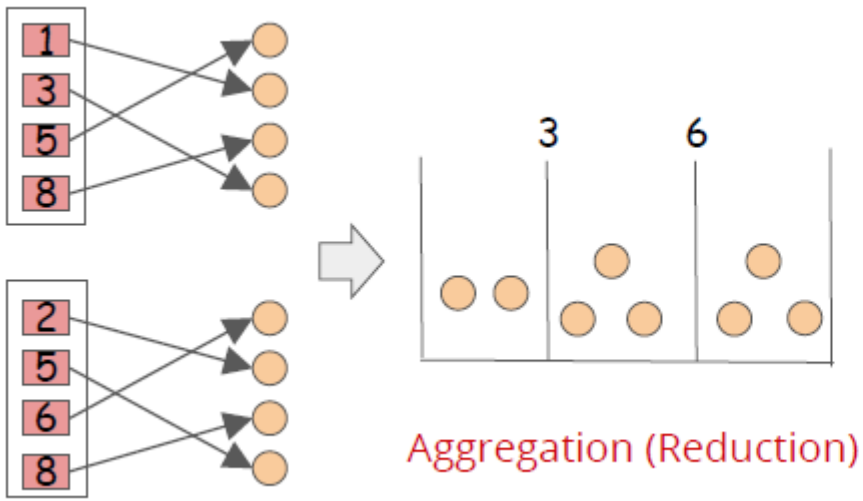
Long range instruction dependency

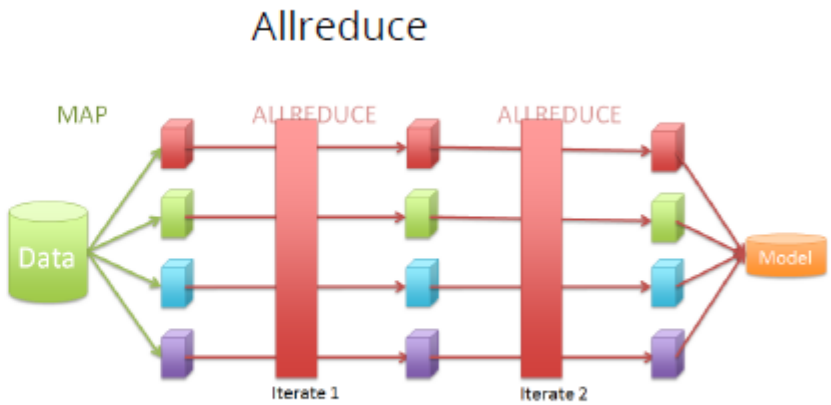
Continuous memory access

Prune

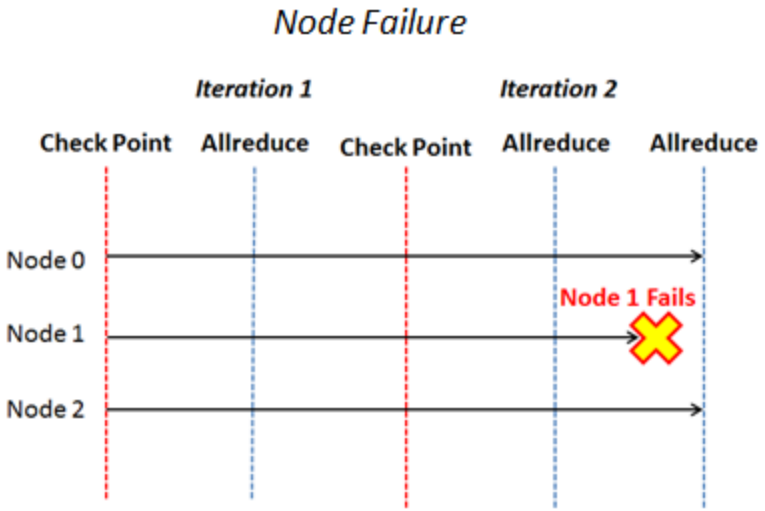
```
updater_prune->Update()
  for (each tree)
    ->DoPrune();
        ->for (each node)
          //if loss_chg<gamma, prune. This is a recursive process
          TryPruneLeaf();
```

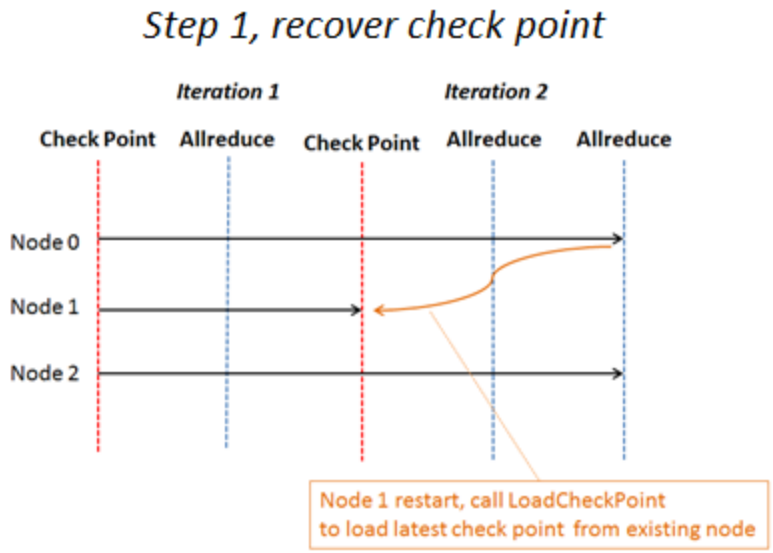
Sketch of Distributed Learning Algorithm

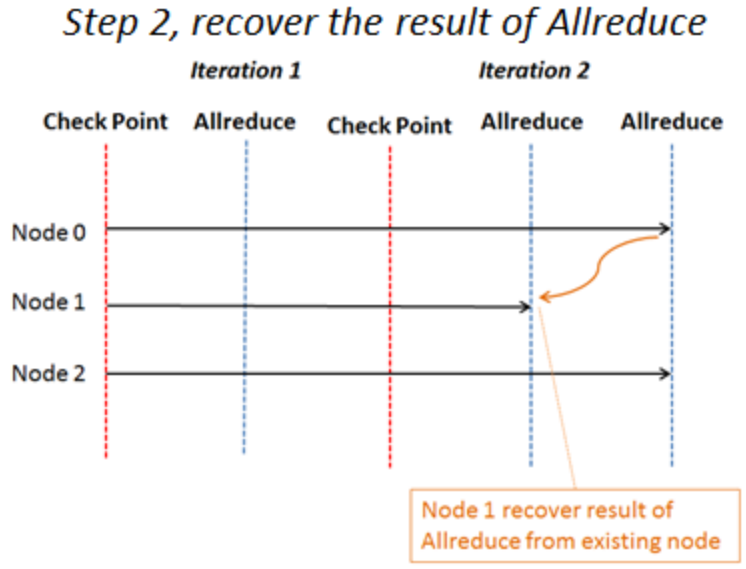


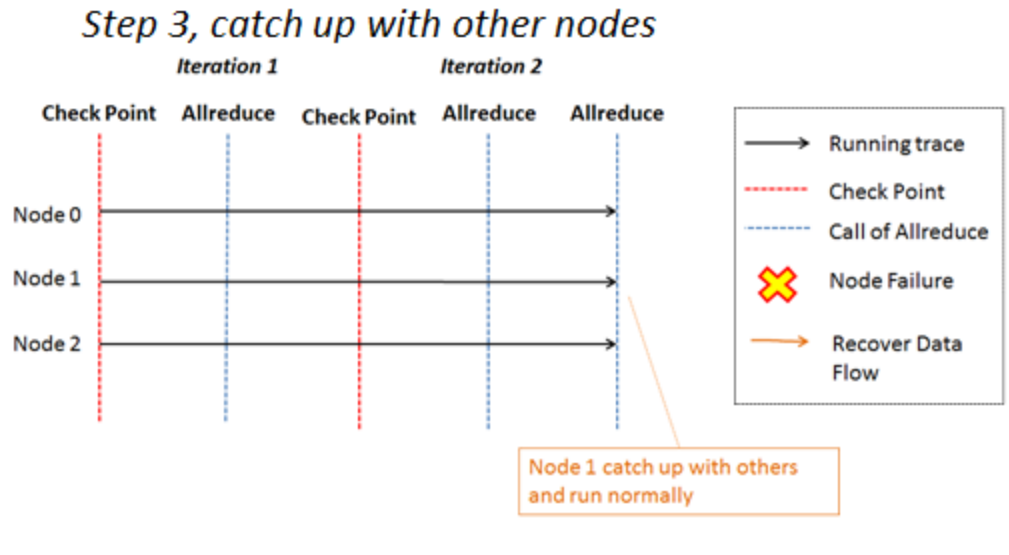


Rabit


Node Failure
Iteration 1
Iteration 2
Check Point
Allreduce
Check Point
Allreduce
Allreduce
Node 0
Node 1
Node 2
Node 1 Fails


Step 1, recover check point
Iteration 1
Iteration 2
Check Point
Allreduce
Check Point
Allreduce
Allreduce
Node 0
Node 1
Node 2
Node 1 restart, call LoadCheckPoint
to load latest check point from existing node


Step 2, recover the result of Allreduce
Iteration 1
Iteration 2
Check Point
Allreduce
Check Point
Allreduce
Allreduce
Node 0
Node 1
Node 2
Node 1 recover result of
Allreduce from existing node


Step 3, catch up with other nodes
Iteration 1
Iteration 2
Check Point
Allreduce
Check Point
Allreduce
Allreduce
Node 0
Node 1
Node 2
Node 1 catch up with others
and run normally
Running trace
Check Point
Call of Allreduce
Node Failure
Recover Data
Flow

Summary

- Combine state-of-art boosting optimization and CS design
- Open source community, keep incorporating new features:
  - col sub sample/ num_parallel_tree (random forest)
  - Dart
  - Fast histogram (LGB)
  - Momentum
  - R/Python/Julia api
- Unix Philosophy in Machine Learning

Reference

- XGBoost: A Scalable Tree Boosting System
- A Fast Algorithm for Approximate Quantiles in High Speed Data Streams
- DART: Dropouts meet Multiple Additive Regression Trees
- McRank: Learning to Rank Using Multiple Classification and Gradient Boosting
- Parallel Boosted Regression Trees for Web Search Ranking
- RABIT: A Reliable Allreduce and Broadcast Interface

- http://mlnote.com/2016/10/05/a-guide-to-xgboost-A-Scalable-Tree-Boosting-System/
- http://mlnote.com/2016/10/29/xgboost-code-review-with-paper/
- http://blog.csdn.net/flydreamforever/article/details/75805924
- http://blog.csdn.net/flydreamforever/article/details/76219727
- http://blog.csdn.net/matrix_zzl/article/details/78699605
- http://blog.csdn.net/matrix_zzl/article/details/78705753
- http://blog.csdn.net/chedan541300521/article/details/54895880
- http://blog.csdn.net/zc02051126/article/details/46711047
- http://blog.csdn.net/wzmsltw/article/details/50994481
- https://www.analyticsvidhya.com/blog/2016/03/complete-guide-parameter-tuning-xgboost-with-codes-python/
- https://stackoverflow.com/questions/43542317/xgboost-early-stopping-cv-versus-gridsearchcv
- https://cosx.org/2015/03/xgboost
- Kaggle Winning Solution Xgboost Algorithm - Learn from Its Author, Tong He
- https://tqchen.github.io/2016/03/10/story-and-lessons-behind-the-evolution-of-xgboost.html
- https://www.zhihu.com/question/41354392
- http://zhanpengfang.github.io/418home.html
- https://github.com/dmlc/rabit/blob/master/doc/guide.md